IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant : Gilbert Wolrich et al.   Art Unit : 2154
Serial No. : 10/069,229   Examiner : Joshua Joo
Filed : December 11, 2002   Conf. No. : 1429

Title : BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE

MAIL STOP APPEAL BRIEF – PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

REPLY BRIEF

Pursuant to 37 CFR § 41.41, Appellant's response to arguments raised in the Examiner's Answer mailed on December 13, 2006, is as follows:

Claims 1, 14, 17 and 20 – The References Teach Away from Their Combination

The Examiner did not address Appellant's argument that the cited references that were used to reject appellant's independent claim 1 teach away from their combination. Appellant continues to rely on this argument, as presented in the Appeal Brief, that the no *prima facie* case of obviousness has been established with respect to independent claims 1, 14, 17 and 20, and the claims that depend from them, for at least the reason that the cited references teach away from their combination.

Claims 1, 14, 17 and 20 – No Motivation to Combine Carron, Yates and White

Responding to appellant's argument that to combine Carron (U.S. Patent No. 4,724,521) with White (U.S. Patent No. 5,748,950) and/or Yates (U.S. Patent No. 5,802,373) would render Carron unsatisfactory for its intended purpose, the Examiner explained his understanding of the Carron reference as follows:

> In response, Carron teaches,
>
> > "the general purpose operation routines will be created in machine language, which can be assembled into machine instructions which, in turn, can be directly executed by the central processor unit in the local

terminal. This central processor unit will typically be a microprocessor having a specific machine instruction set. Utilizing machine language programming and having such programs authored by a highly skilled machine language programmer will provide the fastest execution time for individual general purpose operation routines and thus faster execution time for application programming running the local terminal. (Col 18, lines 40-52).

"the method of this invention is based on storing in read only memory circuits within the local terminal a number of general purpose operation routines which comprise instructions to be executed by the central processor unit to accomplish a particular program task. Each of these general purpose operation routines is associated with a defined command. (Col 7, lines 59-66).

Carron teaches that the routines are processor-executable instructions, in which routines are associated with commands. The commands cause routines to be executed by a processor to accomplish a task, thus Carron's commands does not avoid machine-level instructions as asserted by the appellant. Furthermore, regardless of whether Yates and White's instructions are for particular processing environments, all three references teach similarly of executing instructions for comparing and branching, and the specific instructions of Yates and White would provide an enhancement to the commands taught by Carron. (Examiner's Answer, Page 12)

The Examiner acknowledges that Carron describes commands that cause routines, implemented as sequences of processor-executable instruction, to be executed to accomplish a task. In other words, and as appellant has observed in the Appeal Brief, Carron's COMP_CHAR or COMP_BYTE commands, much like Carron's other commands, are not themselves processor-executable instructions but rather are commands stored in the memory of a terminal executing the COMP_CHAR and/or COMP_BYTE commands, each of which, in turn, causes its corresponding sequence of processor-executable instructions to be executed. Indeed, as Carron explicitly states:

It should be apparent from the above description that the method of this invention makes it possible for the programmer to create application programs with **commands in a source level language** that are particularly suited to the type of application for the terminals that are to be programmed. The commands are compiled into compressed operation codes and parameters which reduce the size of the code which must be downloaded to a practicable size. The methods of this invention provide for parallel execution of APMs which greatly expands the facility with which the programmer can implement the execution of the program tasks in the local terminal without wasting time for inputs to arrive. This increases the efficiency of use of the

> central processor unit within the terminal. **(emphasis added, Carron, col. 105, lines 49-63)**

In Carron explicit language, which is consistent with the Examiner's interpretation, Carron's commands correspond to a high-level programming language commands. Therefore, Carron's high-level language commands are incompatible with architecture-specific operands, such as designations of processor-specific registers, or designations of particular bytes of the processor-specific registers.

In contrast, and as explained in appellant's Appeal Brief, White's COBR instruction (which the examiner relies upon to reject appellant's independent claims) is a machine level instruction that performs a comparison operation using registers, and is configured to be executed on the particular processing environment shown in White's FIG. 2.

Similarly, Yates instruction sequences, including the instruction sequences 884a and 884b shown in FIG. 65B (i.e., "CMPB AL, #3; BNE) are also machine-level instructions configured to be executed in a particular processing environment.

Because Carron requires that commands, including the COMP_CHAR command, be provided in a compact high-level language that does not refer to architecture-specific operands, such as registers, to modify Carron's high-level commands by combining it with White's and/or Yates machine-level instructions would render Carron unsatisfactory for its intended purpose and/or change Carron's principle of operation. Indeed, not only are Carron's high-level commands incompatible with White's and Yates' machine-level instructions, but use of White's and Yates machine-instructions would also preclude Carron's high-level commands from executing on different types of processors.

Because combining White and/or Yates with Carron would render Carron unsatisfactory for its intended purpose and/or would change Carron's principle of operation, no motivation for combining Carron with Yates and/or White exists. The Examiner has thus failed to establish a *prima facie* case of obviousness with respect to independent claim 1, as well as with respect to independent claims 14, 17 and 20. For this reason alone, independent claims 1, 14, 17 and 20, and the claims that depend from them, are therefore patentable over the cited art.

Applicant : Gilbert Wolrich et al.
Serial No. : 10/069,229
Filed : December 11, 2002
Page : 4 of 9

Attorney's Docket No.: 10559-395US1

Claims 4 and 5 -- The Characterization of Atkins Is Incorrect

Responding to Appellant's arguments that the Atkins reference (U.S. Patent No. 5,898,866) does not describe the feature that the branch instruction comprises an optional token that is set by a programmer and specifies a number of instructions i to execute following execution of the branch instruction and before performing the branch operation, the Examiner stated:

> (3) Regarding dependent claim 4, examiner's characterization of Atkins is incorrect. Atkins does not teach or disclose at least, "wherein the branch instruction comprises: an optional token that is set by a programmer and specifies a number i of instructions to execute following the branch instruction before performing the branch operation."
> In response, the examiner stated in the office action dated 3/17/2006, that "Atkins teaches of implementing hardware to execute loops for a branch, wherein a field in the instruction specifies the number of instructions to execute prior to branch." Atkin's specification recites that, "[a] instruction has a length field which specifies the number of instructions within the loop" (Col 2, lines 28-31), and "It is decremented with each iteration to control the branch at the bottom of the loop. BOT is the number of instructions within the loop (i.e. two)." (Col 10, lines 1-3). The examiner's representation of Atkin's is clearly a reasonable interpretation and within the intended meaning of the reference, and thus not a mischaracterization of Atkins. (emphasis added, Examiner's Answer, page 13)

Appellant respectfully disagrees with the Examiner's contentions.

As explained in Appellant's Appeal Brief, the optional token specifies the number of instructions that are to be executed following (i.e., after) execution of the branch instruction. In contrast, Atkins does not describe a branch instruction, and for that reason alone Atkins cannot be said to disclose or suggest "an optional token that is set by a programmer and specifies a number of instructions i to execute following the branch instruction before performing the branch operation."

Rather, Atkins describes a SETLOOP instruction that initializes a loop operation. The SETLOOP instruction has a length field value, stored in a counter, that is decremented until the branch at the bottom of loop is reached, at which point another iteration of the loop controlled by the SETLOOP instruction may begin. Thus, if the Examiner is interpreting the branch operation performed at the end of a loop iteration as a branch instruction, then under these circumstances the effect of Atkins' length field value is to specify the number of instructions that are to be

executed *before* execution of the loop's branch. Accordingly, even under this interpretation Atkins' SETLOOP instruction and its associated length field parameter do not correspond to a branch instruction that comprises "an optional token that is set by a programmer and specifies a number of instructions i to execute following the branch instruction before performing the branch operation."

Appellant therefore contends that the examiner's characterization of Atkins is incorrect. Appellant's claims 4 and 5 are therefore patentable over the cited art.

Claims 7 and 11 – The Characterization of Bruckert Is Incorrect

Responding to appellant's arguments that the Bruckert reference (U.S. Patent No. 4,742,151) does not describe the feature that the branch instruction comprises an optional token that is set by the programmer and which specifies a guess_branch prefetch for the instruction for the "branch taken" condition rather than the sequential instruction, the Examiner stated:

> (4) Regarding dependent claim 7, Brucket describes a processor that fetches two instructions when the processor encounters a branch instruction: one instruction corresponding to the "branch taken" eventuality as well as the "branch not taken". Neither of these eventualities correspond to the claimed branch guess token. Because both instructions are retrieved, Brucket would have no need to include a token that would indicate which instruction should be retrieved following the execution of a branch instruction.
> In response, Brucket teaches of determining whether a branch should be taken depending on prior processing and prefetching the "branch taken" instruction and "branch not taken" instruction (Col 1, lines 22-29). Brucket does not explicitly teach of a "token" for specifying a prefetch for the "branch taken" instruction. However, an indicator or an identifier such as a token would still be needed to indicate the retrieval of the "branch taken" instruction. Furthermore, the prefetch for "branch taken" or the indication for prefetch, e.g. indicator, identifier, or token, is optional in that a programmer has the option to include the specific prefetch as part of the programming. Appellant's "optional token" in the claim does not define what is optional about the token such that the token is selected over another token. (Examiner's Answer, pages 14-15)

Appellant's respectfully disagrees with the Examiner's contentions.

Firstly, with respect to the recitation "optional token", the adjective "optional" qualifies the token. In other words, a programmer may, in some circumstances, decide to use the

Applicant : Gilbert Wolrich et al.
Serial No. : 10/069,229
Filed : December 11, 2002
Page : 6 of 9

Attorney's Docket No.. 10559-305US1

branch_guess token, and in some other circumstances not to use the branch_guess token. The "optional", therefore, refers to the option a programmer has on whether or not to use the token.

As for the alleged inclusion of a token in Bruckert's instructions, Bruckert describes:

> **The invention enables the processor to determine if an instruction is a conditional branch instruction, in which a determination is made as to whether a branch in an instruction stream should or should not be taken depending on the results of prior processing, and to prefetch both the instructions in the "branch taken" instruction stream as well as from the "branch not taken" instruction stream. (Bruckert, col. 1, lines 22-29)**

Bruckert, as explained in Appellant's Appeal Brief, does not include an instruction, or a parameter of an instruction, that would specify which of two possible target instructions (or instruction streams) should be retrieved (i.e., the instructions immediately following the branch instruction, or the instructions at the target branch address presumably specified by the branch instruction.) Rather, Bruckert retrieves both instruction streams. Because Bruckert retrieves both instruction streams, Bruckert has no need to include a token that would indicate which instruction stream should be retrieved following the execution of a branch instruction.

The Examiner's characterization of Bruckert is therefore wrong, and Bruckert neither discloses nor suggests at least "an optional token that is set by a programmer and which specifies a guess_branch prefetch for the instruction for the "branch taken" condition rather than the next sequential instruction," as required by Appellant's claims 7 and 11.

Claim 8 — The Characterization of Carron and Bruckert Is Incorrect

Responding to Appellant's arguments as set forth in the Appeal Brief, the Examiner stated:

> (5) Regarding dependent claim 8, Carron does not disclose or suggest an optional token that specifies a number i of instructions to execute following a branch instruction and a second optional token that is set by a programmer and which specifies a guess_branch prefetch for the instruction for the "branch taken" condition rather than the next sequential instruction. Carron does not disclose or suggest anywhere that its COMP_BYTE command performs any type of branch defer operation that causes the execution of additional instructions following the branch instruction and before the branch operation is performed.
> In response, firstly, claim 8 does not specifically define the argued feature of a "branch defer operation". Therefore, it is not clear as to what appellant's argued "branch defer operation" is specifically

referring to in the appeal brief. It appears appellant is equating the
branch defer operation with the "guess_branch prefetch" and will be
interpreted as such. Claim 8 also does not define a feature of performing
a branch defer operation that causes the execution of additional
instructions following the branch instruction and before the branch
operation. Claim 8 recites, in additional to other features, "a second
optional token... which specifies a guess_branch prefetch for the
instruction for the "branch taken"," but the claim does not define the
feature that the guess_branch prefetch causes the execution of
additional instructions nor that the guess_branch prefetch is following the
branch instruction and before
the branch operation. It is noted that the features upon which appellant
relies are not recited in the rejected claim(s). Although the claims are
interpreted in light of the specification, limitations from the specification
are not read into the claims. See In re Van Geuns, 988 F.2d 1181, 26
USPQ2d 1057 (Fed. Cir. 1993). (emphasis added, Examiner's Answer,
page 15)

Appellant respectfully disagrees with the Examiner's contentions.

Appellant's claim 8, as provided in the originally filed application, and as provided in the
appendix to the Appeal Brief, recites:

> 8. The instruction of claim 1 wherein the branch instruction
> comprises:
>   an optional token that is set by a programmer and specifies a
> number i of instructions to execute following the branch
> instruction before performing the branch operation; and
>   a second optional token that is set by a programmer and which
> specifies a guess_branch prefetch for the instruction for the
> "branch taken" condition rather than the next sequential
> instruction.

Appellant's claim 8 includes the recited feature of "an optional token ... specifies a
number i of instructions to execute following the branch instruction before performing the
branch operation." Appellant in arguing claim 8 merely referred to this function by the short
hand expression of "a branch defer operation." which corresponds to the recited feature of an
optional token ... specifies a number i of instructions to execute following the branch instruction
before performing the branch operation.

The Examiner next stated in the Examiner's Answer:

Carron teaches of the COMP_BYTE command or the COMP_CHAR command, commands that invoke instructions for executing a branch operation (Col 134, lines 14-26, 37-49), to which appellant also stated in appellant's Remarks dated 12/29/2005, page 9, "Carron's COMP_BYTE, much like Carron's other commands... invokes a routine comprising several instructions stored in memory." Therefore, when creating the command, a programmer may specify the number of instructions, i.e. how many or which instructions, to execute prior to the branch operation. Carron further teaches of a command comprising, "Get the number of instructions on the application input circular list." and "Decrement the number of instructions remaining." (Col 592, lines 32-35, 48-52) (Examiner's Answer, page 16)

While Carron's commands are constituted by a sequence of instructions, and while Carron may include a command that, for argument's sake, may teach "[g]et the number of instructions on the application input circular list" and "[d]ecrement the number of instructions remaining," none of these discloses or suggests a branch defer operation in which a token, of a branch instruction, specifies the number of instruction to execute following the execution of a branch instruction.

Lastly, the Examiner stated:

As set forth in the office action dated 3/17/2006, Carron does not specifically teach, "a second optional token that is set by a programmer and which specifies a guess_branch prefetch for the instruction for the "branch taken" condition rather than the next sequential instruction. Brucket teaches of determining whether a branch should be taken depending on prior processing and prefetching the "branch taken" instruction and "branch not taken" instruction (Col 1, lines 22-29). Brucket does not explicitly teach of a "token" for specifying a prefetch for the "branch taken" instruction. However, a command or an identifier such as a token would be needed to indicate the prefetch for the "branch taken" instruction. Furthermore, the prefetch for "branch taken" or the indication for the prefetch, e.g. command, identifier, or token, is optional in that a programmer has the option to include the specific prefetch as part of the programming. Appellant's feature of optional token in the claim does not define what is optional about the token. (Examiner's Answer, page 16)

For reasons similar to those provided above with respect to claims 7 and 11, Appellant submits that this feature is not disclosed or suggested by Bruckert. Claim 8 is therefore patentable over the cited art.

Applicant : Gilbert Wolrich et al.
Serial No. : 30/069,229
Filed : December 11, 2002
Page : 9 of 9

Attorney's Docket No.: 10559-305US1

Conclusion

For the foregoing reasons, and the reasons stated in the Appeal Brief, appellant submits that the final rejection should be reversed.

Please apply any required fees or credits to deposit account 06-1050, referencing the attorney docket number shown above.

Respectfully submitted,

Date: Jan 30, 2007

Ido Rabinovitch
Attorney for Intel Corporation
Reg. No. L0080

PTO Customer No. 26161
Fish & Richardson P.C.
Telephone: (617) 542-5070
Facsimile: (617) 542-8906

21546625.doc